Core Features of React for a Full-Fledged Developer

# 1. React Fundamentals

* • JSX - HTML-in-JS syntax used in React components.
* • Components - Reusable building blocks — Functional & Class components.
* • Props - Data passed to components (read-only).
* • State - Internal, mutable data used for interactivity.
* • Rendering - Conditional rendering, list rendering (.map).
* • Event Handling - Handling events like onClick, onChange, etc.
* • Fragments & Short Syntax - <> </> for grouping without extra DOM nodes.
* • React.StrictMode - Tool to highlight potential issues in development.

# 2. Component Lifecycle & Side Effects

* • useEffect Hook - Handle side effects like API calls, DOM access.
* • Mounting / Unmounting - React lifecycle flow understanding.
* • Cleanup in useEffect - Prevent memory leaks and cleanup tasks.

# 3. React Hooks (Modern Essentials)

* • useState - Declare state in functional components.
* • useEffect - Side effects like fetching or timers.
* • useContext - Share global state without prop drilling.
* • useRef - Access DOM or persist values without re-render.
* • useMemo - Memoize values to avoid expensive re-calculations.
* • useCallback - Memoize functions to avoid unnecessary re-renders.
* • useReducer - Alternative to useState for complex state logic.
* • Custom Hooks - Extract and reuse logic across components.

# 4. Routing with React Router

* • BrowserRouter, Routes, Route - Define client-side routes.
* • Dynamic Routing - /user/:id, etc.
* • useParams, useNavigate - Read route params and navigate programmatically.
* • Nested Routes - Hierarchical component routing.
* • Protected Routes - Authentication-based route access.

# 5. State Management (Global)

* • React Context API - Share global state simply.
* • Redux Toolkit - Powerful predictable state container (standard in complex apps).
* • Zustand / Jotai / Recoil - Simpler state managers (modern alternatives).

# 6. Forms and Input Handling

* • Controlled Components - Sync form values with state.
* • Uncontrolled Components - Using refs for form handling.
* • Form Validation - Manual or using libraries like react-hook-form, Formik, Yup.

# 7. API Integration

* • Fetch / Axios - Call external APIs.
* • Loading and Error States - UX feedback during async operations.
* • Custom Hooks for API - Encapsulate fetching logic.

# 8. Component Styling

* • CSS / SCSS - Traditional styling.
* • CSS Modules - Scoped styles.
* • Tailwind CSS - Utility-first CSS framework (popular).
* • Styled-Components - CSS-in-JS styling.
* • DaisyUI / Chakra UI - Ready-made UI kits for React.

# 9. Code Structure & Architecture

* • Reusable Component Design - DRY principles, props usage.
* • Folder Structure - Organize by feature or type.
* • Separation of Concerns - Keep logic, views, and data separate.

# 10. Developer Tooling

* • Create React App / Vite - Boilerplate setup.
* • React DevTools - Debug state, props, etc.
* • ESLint & Prettier - Code linting and formatting.
* • VS Code Extensions - React Snippets, GitLens, etc.
* • Version Control (Git) - Essential for collaboration.

# 11. Testing in React

* • Unit Tests - Jest.
* • Component Tests - React Testing Library.
* • End-to-End Tests (E2E) - Cypress / Playwright.

# 12. Deployment

* • Netlify / Vercel - Fast static hosting (CI/CD built-in).
* • Firebase Hosting - Hosting + database + auth.
* • GitHub Pages - Free basic hosting.

# 13. Advanced Concepts (For Mastery)

* • Code Splitting / Lazy Loading - Faster load time with React.lazy.
* • Error Boundaries - Catch component errors gracefully.
* • SSR / SSG (Next.js) - Server-side rendering for SEO and performance.
* • PWA support - Offline capability, installable apps.
* • Accessibility (a11y) - ARIA, keyboard navigation, screen readers.
* • Type Safety - TypeScript with React.